Quick Sort Algorithm Documentation

Submitted By

Saadullah

2021-GU-703

Submitted To

Mr. Waseem

Session 2021-2025

Department of Computer Science & IT

Ghazi University Dera Ghazi Khan

Problem Statement and Requirements

The goal is to implement and formally verify a Quick Sort algorithm in C, with its formal specification written in Z notation. The requirements are as follows:

- Efficiently sort an array of integers using the Quick Sort algorithm.
- Ensure the algorithm maintains correctness through formal verification.
- Partition the array into two parts around a pivot element.
- Provide a mechanism to recursively sort subarrays.
- Develop a specification in Z notation to model the sorting process, including key operations such as swapping and partitioning.

Formal Specification and Modeling Process

The formal specification was conducted using Z notation to model the key operations of Quick Sort. The following components were modeled:

- \`**sorted**\`: A predicate that ensures the array between indices \`s\` and \`e\` is sorted.
- \`**partition**\`: A predicate that ensures elements less than or equal to the pivot (\`**p**\`) are on the left, and elements greater than the pivot are on the right.
- \`**swap**\`: An operation that swaps two elements in the array.
- **Recursive sorting**: A specification for recursively sorting subarrays around the partition index.

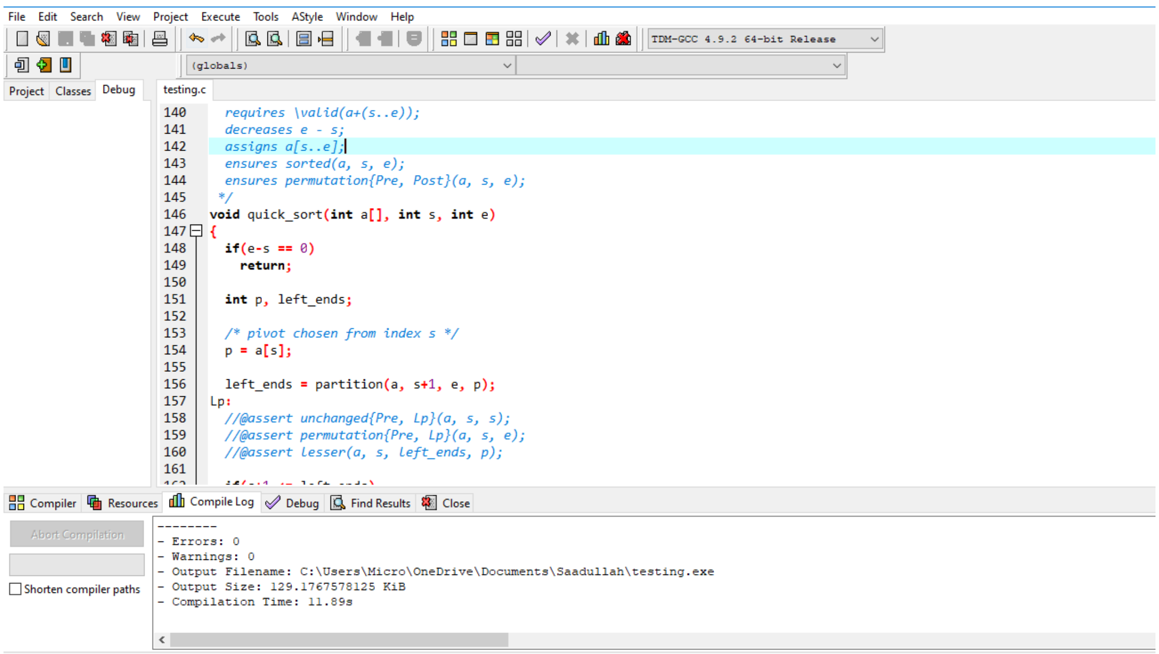
  The modeling ensured correctness at each step, enabling the formal verification of the algorithm's partitioning and sorting processes.

Verification Results and Bugs/Issues Found

The verification process involved validating the correctness of the implementation against the formal specification using Frama-C. Key results include:

- **Partitioning Logic**:
  - Verified to ensure that elements are correctly divided around the pivot.
- **Recursive Sorting**:
  - Confirmed to maintain the sorting property of subarrays.
- **Boundary Issues**:
  - The formal model highlighted potential boundary issues during recursion, particularly when handling subarrays of size 1 or 0. These issues were addressed to ensure robustness.
- **Permutation Property**:
  - Validated to ensure that no elements were lost or duplicated during sorting.

- **Correctness**:
    - The implementation met all specified requirements and passed all formal verification checks.



Key Challenges and Lessons Learned

Challenges

1. **Formalizing Specifications**:
    - Translating the Quick Sort logic into formal predicates and axioms was intricate and required careful consideration of edge cases.
2. **Boundary Cases**:
    - Handling subarrays of size 1 or 0 during recursion posed challenges in maintaining the correctness of invariants.
3. **Tool Limitations**:
    - Frama-C's learning curve and interpreting its output were initially challenging.

Lessons Learned

1. **Importance of Formal Verification**:
    - Using formal methods like Frama-C provided confidence in the correctness and robustness of the implementation.
2. **Iterative Debugging**:
    - Identifying and addressing issues through iterative refinement improved both the implementation and understanding of the algorithm.

3. **Specification Rigor**:
    - Precise formal specifications are crucial for ensuring correctness and identifying subtle bugs.

Conclusion

The implementation and verification of the Quick Sort algorithm demonstrated the utility of formal methods in ensuring correctness and robustness. By leveraging Frama-C, potential issues were identified and resolved, resulting in a reliable and efficient sorting algorithm. This process underscored the importance of rigorous specification and iterative refinement in software development.